Draft for Comments

CHAPTER: VISUAL PROCESSING

Text Book on
Fundamentals of Multimedia Computing

Authors:
Gerald Friedland
and
Ramesh Jain

Draft for Comments

Chapter: Visual Processing

Introduction

Vision is the most powerful human sense. It is commonly believed that a significant part of human perceptual processing, almost two-thirds, is devoted to visual perception. Vision is so important because it provides overview of spatio-temporal information both at overview level as well as local level. Moreover, this is the only sense that can be easily tuned to change its scope to acquire information at different levels of granularity. Because of the generality of vision, humans have a tendency to convert all types of information to visual form for understanding it better. The most interesting example is how text evolved. Text is a visual representation of speech, which is a small fraction of sounds that we understand and recognize. Thus text is an effort to represent sound in visual form. Our language is full of metaphors related to vision; the most common being statements like: Do you *see* my point?

Two common representations of visual information are images (photos) and image sequences (video). A photo is a frozen representation of a visual 'moment' from a specific perspective. A photo provides us spatial layout of optical characteristics resulting from the interaction of optical characteristics of objects and different light sources. A video captures the dynamic happenings from a particular viewpoint. Thus, video is a very rich spatio-temporal representation of optical characteristics due to interactions among objects in an environment.

In many applications of multimedia in early days, video was considered more or less equivalent to multimedia. As the field evolved, it became clear that video is a rich source of information, but unless it is combined with other sources of information, it may be very difficult to recover any information from it. Some important aspects of correlation among different sources and their role in understanding are discussed in a chapter on <Context>.

Visual Information

Many fields in computer sciences address visual processing. We briefly discuss some very closely related fields to provide a general idea of the topics addressed in computer sciences and related areas concerned with some aspects of visual processing of information. The fields most closely related to visual information processing are commonly called computer vision and machine vision. In this section we will refer to them as vision systems or just vision. The main goal of these systems is to extract relevant information from images or video.

Image processing techniques usually transform images into other images; the task of information recovery is left to a human user. Image processing field studies topics such as image enhancement, image compression, and correcting blurred or out-of-focus images. In general, it studies the process of image formation using different devices and how to improve quality of those images to make them closer to the physical phenomena they represent. In some cases, the goal is to enhance the objects they represent. *Machine vision* algorithms take images as inputs but produce other types of outputs, such as representations for the object contours in an image. Thus, emphasis in machine vision is on recovering information automatically, with minimal interaction with a human. Image processing algorithms are useful in early stages of a machine vision system. They are usually used to enhance particular information and suppress noise.

Computer graphics generates images for a given model using geometric primitives such as lines, circles, and free-form surfaces. Computer graphics techniques play a significant role in visualization and virtual reality. Machine vision is the inverse problem: estimating the geometric primitives and other features from the image. Thus, computer graphics is the synthesis or creation of images, and machine vision is the analysis of images. In the early days of these two fields, there was not much relationship between them, but in the last two decades these two fields have been growing closer. Machine vision is using curve and surface representations and several other techniques from computer graphics, and computer graphics is using many techniques from machine vision to enter models into the computer for creating realistic images. Visualization and virtual reality are bringing these two fields closer.

Pattern recognition classifies numerical and symbolic data using models, generally statistical models, for recognition of objects. Many statistical and syntactical techniques have been developed for classification of patterns for various applications. Techniques from pattern recognition play an important role in machine vision for recognizing objects. In fact, many machine vision techniques rely on *machine learning* algorithms to learn particular object or concept models and then to recognize them. Machine learning techniques are direct derivative of pattern recognition approaches. In pattern recognition the models for objects were assumed to be available, while machine learning assumes that enough data sets are available so that using computational approaches, the system can develop models for the objects to be recognized and then use these models for classifying objects.

Artificial intelligence is concerned with designing systems that are intelligent and with studying computational aspects of intelligence. Artificial intelligence is used to analyze scenes by computing a symbolic representation of the scene contents after the images have been processed to obtain features. Artificial intelligence may be viewed as having three stages: perception, cognition, and action. Perception translates signals from the world into symbols, cognition manipulates symbols, and action translates symbols into signals that effect changes in the world. Many techniques from artificial intelligence may play important roles in all aspects of visual information processing.

Psychophysics, along with cognitive science, has studied human vision for a long time. Many techniques in machine vision are inspired by what is known about human vision. In fact, many researchers in computer vision are more interested in preparing computational models of human vision.

It may be useful to remember the equation

$$\text{Vision} = \text{Geometry} + \text{Measurements} + \text{Interpretation}$$

Thus, vision comprises techniques for estimating features in images, relating feature measurements to the geometry of objects in space, and interpreting this geometric information.

Role of Knowledge

Decision-making usually requires knowledge of the application or goal. Emphasis in vision systems on maximizing automatic operation means that these systems should use knowledge to accomplish this. The knowledge used by the system includes models of features, image formation, models of objects, and relationships among objects. Without explicit use of knowledge, vision systems can be designed to work only in a very constrained environment for very limited applications. To provide more flexibility and robustness, knowledge is represented explicitly and used by the system. In many cases, such knowledge is made available from the context in which these systems work. In Chapter <Context> we will discuss how such models could really be used effectively.

Image Geometry

There are two parts to the image formation process:

- The geometry of image formation, which determines where in the image plane the projection of a point in the scene will be located.
- The physics of light, which determines the brightness of a point in the image plane as a function of scene illumination and surface properties.

Although an understanding of the physics of light is not necessary for understanding the fundamentals of most vision algorithms, such knowledge is useful in building vision systems. Since in our context, we are not concerned with detailed design consideration of vision systems, we will not discuss those processes in details here. The geometry of

image formation uses the basic model for the projection of points in the scene onto the image plane as diagrammed in Figure 1. In this model, the imaging system's center of projection coincides with the origin of the three-dimensional coordinate system. The coordinate system for points in the scene is the three dimensional space spanned by the unit vectors **x**, **y**, and **z** that form the axes.

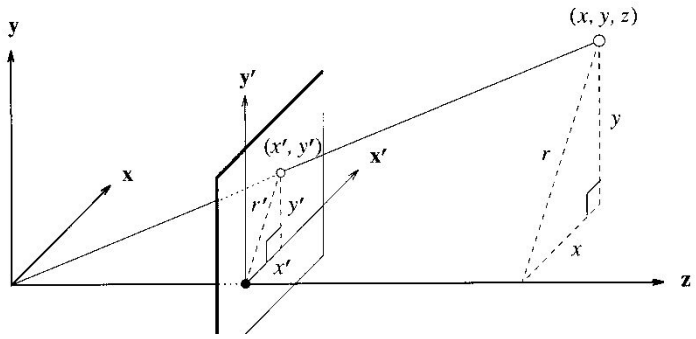


Figure 1 The point on the image plane that corresponds to a particular point in the scene is found by following the line that passes through the scene point and the center of projection of the coordinate system.

Deleted: <Figure to be redrawn.>

A point in the scene has coordinates (x, y, z) and its projection in the camera image appears at location (x', y'). The *x* coordinate is the horizontal position of the point in space as seen from the camera, the *y* coordinate is the vertical position of the point in space as seen from the camera, and the *z* coordinate is the distance from the camera to the point in space along a line parallel to the z axis. The *line of sight* of a point in the scene is the line that passes through the point of interest and the center of projection. The line drawn in Figure 1 is a line of sight.

The image plane is parallel to the x and y axes of the coordinate system at a distance / from the center of projection, as shown in Figure 1. This figure helps us understand how a camera converts a three-dimensional world to a two-dimensional image by essentially a many-to-one mapping. This means that without the knowledge about the domain and the context, an image cannot be unambiguously interpreted. This is the fundamental reason behind the complexity in visual processing.

Sampling and Quantization

A continuous function must be converted to a discrete form for representation and processing using a digital computer. The interface between the optical system that projects a scene onto the image plane and the computer must sample the image at a finite number of points and represent each sample within the finite word size of the computer.

This requires applying sampling and quantization steps. Each image sample is called a pixel. We will assume that images are sampled on a regular grid of squares so that the horizontal and vertical distances between pixels are the same throughout the image. Many cameras acquire an analog image, which is then sampled and quantized to convert it to a digital image. The sampling rate determines how many pixels the digital image will have (the image resolution), and quantization determines how many intensity levels will be used to represent the intensity value at each sample point. The image processing books (see further reading section) discuss the factors that should be considered in selecting appropriate sampling and quantization rates to retain the important information in images. In most modern cameras, the sampling and quantizing rates are predetermined and specified as number of pixels (as MP, or million pixels) available on the chip used in the camera. In this case, the images are directly acquired in digital form.

Levels of Computation

An image usually contains several objects. A vision application usually involves computing certain properties of an object, not the image as a whole. To compute properties of an object, individual objects must first be identified as separate objects; then object properties can be computed by applying calculations to the separate objects. For considering computational aspects of vision algorithms, it helps to consider each algorithm in terms of its input-output characteristics. By considering nature and the level of these operations it is possible to consider how best to implement these operations. Note that the input to a computer vision system is an image, and the output, unlike that of image processing systems, is some symbolic quantity denoting identity or location of an object, for instance. The amount of data processed by a vision system is very large, and that makes the computational requirements very demanding. Since we want to discuss characteristics of operations to predict their computational requirements, we classify the levels of operations and study their general characteristics.

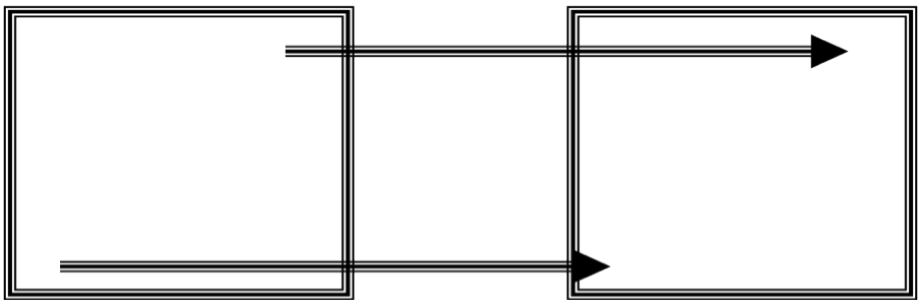
Point Level

Some operations produce an output based on only a point in an image. Thresholding is an example of a point operation. A thresholding algorithm produces output values that depend only on the input value, for a preset threshold. Thus, if I(x,y), and O(x,y) are input and output images, respectively, then for a point operation,

$$O(x,y) = f\,(I\,(x+k,y+l))$$

Meaning that in the output image, the attribute or intensity value at a point depends only on the attribute value at a corresponding point.

This operation can be efficiently implemented using a lookup table.



Formatted: Font color: Auto

Figure 2. *Top:* Point operations are applied to individual image pixels and produce an output image as the result. *Bottom left:*

Local Level

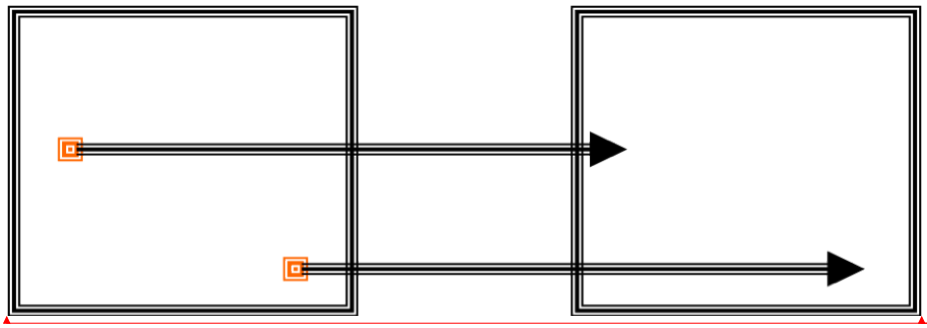
A local operation produces an output image in which the intensity at a point depends on the neighborhood of the corresponding point in the input image.

Thus, for a local operation,

$$O(x,y) = f\,(I\,[Nbr(x,y)])$$

Where Nbr[x,y] denotes a neighborhood of point (x,y). Most commonly used neighborhoods are 3 X 3 windows centered around the point. It is possible to use larger size windows depending on the operation under consideration.

An example of such an operation is shown in Figure 3. Smoothing and edge detection are local operations. Since these operations require values from a neighborhood in the input image, array processors or Single Instruction, Multiple Data (SIMD) machines may be suitable for implementing these operations. In general, these operations can be easily implemented on parallel machines and can often be performed in real time.



Formatted: Font color: Auto

Formatted: Font color: Auto

Figure 3. *Top:* Local operations are applied to pixel neighborhoods and produce an output image as the result.

Global Level

The output of certain operators depends on the whole picture. Such operations are called global operations:

$$P = f(I(x,y))$$

This operation is shown in Figure **4.** The output of these operators may be an image or it may be symbolic output. A histogram of intensity values and the Fourier transform are global operations. We will see that most operations at higher levels are global in nature and are most expensive in terms of time requirements.

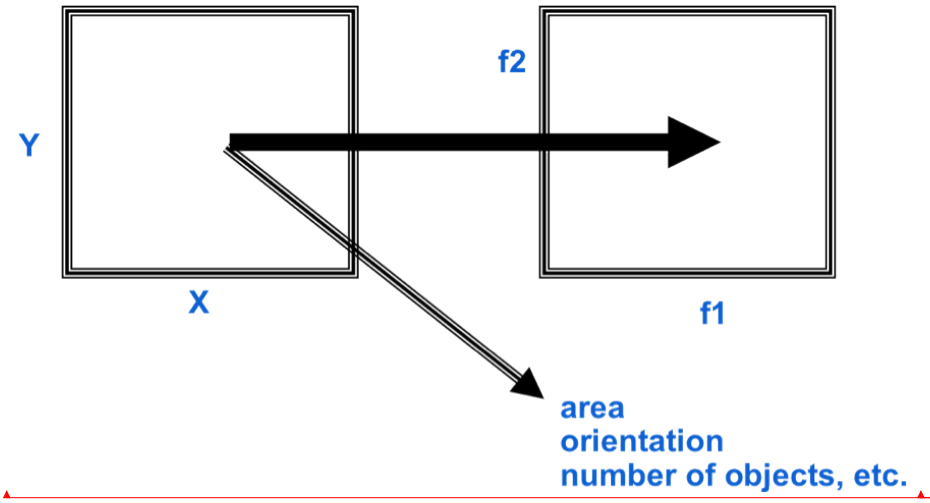


Figure 4. An example of a global operation: an image (left) and its histogram (right). A histogram is a plot of the number of pixels at each gray value contained in the image.

Object Level

Most applications of computer vision require properties to be computed at the object level. Size, average intensity, shape, and other characteristics of an object must be computed for the system to recognize it. Many other characteristics of an object must be determined for detecting features for recognition of an object. This leads to an interesting, but very difficult questions: What is an object? How do we find objects? In Figure 5, how many objects do we see? Some people say 5, some 6 and some may even say 7. All of them are right. We will see that an object is defined in a particular context. In fact, many operations in vision are performed to find where a particular object is

Formatted: Font color: Auto

Formatted: Font color: Auto

located in an image. Objects in images pose a *catch-22* situation. We must use all points that belong to an object to compute some of its characteristics, but we must use those characteristics to identify those points. We will see that significant efforts are spent to solve the *figure-ground* problem (separation of foreground pixels from background pixels) to group points into objects. To understand the contents of an image, a vision system must perform several operations at the object level.

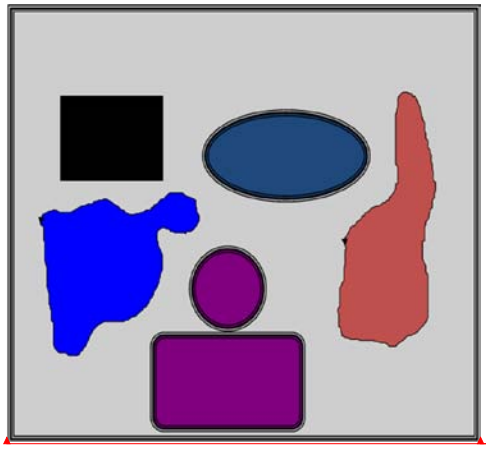


Figure 5. How many objects do we see in this figure? Depending on your definition of an object you may see 5, 6, or 7 objects in this figure.

Formatted: Font color: Auto

Formatted: Font color: Auto

Formatted: Font color: Auto

Thresholding

One of the most important problems in a vision system is to identify the sub-images that represent objects. This operation, which is so natural and so easy for people, is surprisingly difficult for computers. Let us consider that we want to mark all points in an image that belong to an object of interest, commonly called foreground, as 1 and all other points as 0. Thus we convert an image to a binary image that gives us a mask for all object points. Such a binary image is obtained using an appropriate segmentation of a gray scale image. If the intensity values of an object are in an interval and the intensity values of the background pixels are outside this interval, a binary image can be obtained using a thresholding operation that sets the points in that interval to 1 and points outside that range to 0. Thresholding is a method to convert a gray scale image into a binary image so that objects of interest are separated from the background. For thresholding to be effective in *object-background separation*, it is necessary that the objects and background have sufficient contrast and that we know the intensity levels of either the objects or the background. In a fixed thresholding scheme, these intensity characteristics determine the value of the threshold.

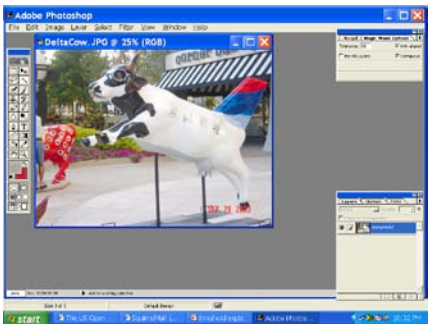
Let us assume that a binary image *B(i,j)* is the same as a thresholded gray image $F_{Thr}[i, j]$ which is obtained using a threshold *T* for the original gray image F[i, j]. Thus,

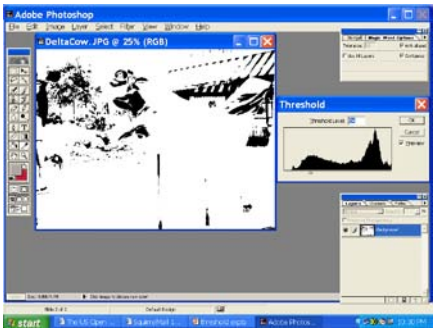
$$B[i, j] = 1 \text{ if } F[i, j] > T$$
$$= 0 \text{ otherwise.}$$

If it is known that the object intensity values are in a range [T1, T2], then we may use

$$B[i,j] = 1 \text{ if } T1 < F[i, j] < T2$$
$$= 0 \text{ otherwise.}$$

The results of producing an image using different thresholds are shown in Figure 6. Note how knowledge about the application domain is required in selecting the threshold. The same threshold values may not work in a new set of images acquired under different conditions. The threshold is usually selected on the basis of experience with the application domain. In some cases, the first few runs of the system may be used for interactively analyzing a scene and determining an appropriate value for the threshold. Automatic thresholding of images is often the first step in the analysis of images in machine vision systems. Many techniques have been developed for utilizing the intensity distribution in an image and the knowledge about the objects of interest for selecting a proper threshold value automatically.





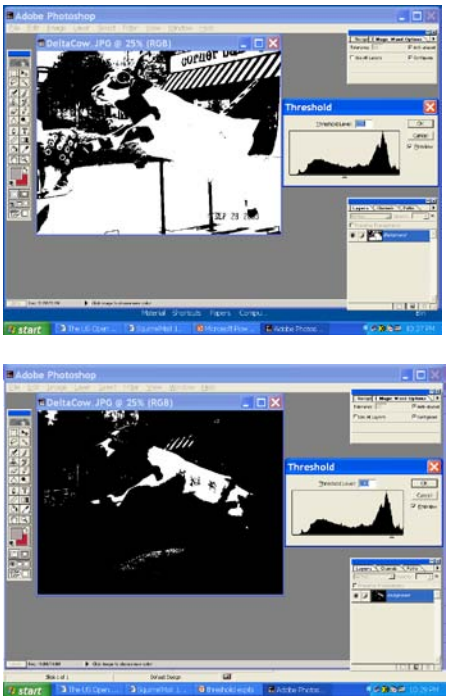


Figure 6. An image and several binary images obtained at different threshold values.

Geometric Properties

Suppose that a thresholding scheme has given us objects in an image. The next step is to recognize and locate objects. In most industrial applications, the camera location and the environment are known. Using simple geometry, one may find the three-dimensional locations of objects from their two dimensional positions in images. Moreover, in most applications the number of different objects is not large. If the objects are different in size and shape, the size and shape features of objects may be determined from their images to help the system recognize them. Many applications utilize some simple features of regions for determining the locations of objects and for recognizing them (e.g., size, position, orientation). If there are several objects, one can compute these features for each object. A connected component or a region usually represents an object. The concept of connectedness and the algorithms for finding connected components in an image will be discussed later in this chapter.

Size

In general, for a binary image it is well known that the area *A* is given by the number of pixels in the image of the object.

Position

The position of an object in an image plays an important role in many applications. There are different ways to specify the position of an object, such as using its enclosing rectangle or centroid. In industrial applications, objects usually appear on a known surface, such as a table, and the position of the camera is known with respect to the table. In such cases, an object's position in the image determines its spatial location. The position of an object in an image may be defined using the center of area of the object image. Though other methods such as the enclosing rectangle of the object image may be used, the center of area is a point and is relatively insensitive to noise in the image.

Binary Algorithms

Segmenting object pixels from background pixels is a difficult problem. We will not address this problem here. Let us assume here that somehow an object can be defined and, using a predicate, the points of an image belonging-to an object may be labeled. The problem then is to group together all points of an image that are labeled as object points into an object image. In this chapter we will assume that all such points are spatially close. This notion of *spatial proximity* requires a more precise definition so that an algorithm may be devised to group spatially close points into a component. For this purpose, let us introduce some definitions.

Neighbors

A pixel in a digital image is spatially close to several other pixels. In a digital image represented on a square grid, a pixel has a common boundary with four pixels and shares a corner with four additional pixels. We say that two pixels are 4 -neighbors if they share a common boundary. Similarly, two pixels are 8-neighbors if they share at least one corner. For example, the pixel at location [i,j] has 4-neighbors [i + 1,j], [i -1,j], [i,j + 1], and [i,j -1]. The 8-neighbors of the pixel include the 4-neighbors plus [i + 1, j + 1], [i + 1, j - 1], [i-1, j + 1] and [i -1, j -1]. A pixel is said to be 4-connected to its 4-neighbors and 8-connected to its 8-neighbors (see Figure 7).

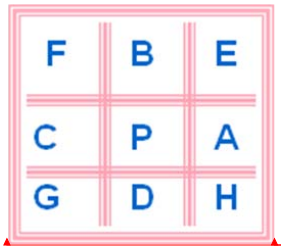


Figure 7. The 4- and 8-neighborhoods for a rectangular image tessellation. Pixel [i, j] is located in the center of each figure. For point P, its 4-neighbors are A, B, C, and D, and it's 8-neighbors will additionally include E, F, G, and H.

Path

A path from the pixel at $[i_0, j_0]$ to the pixel at $[i_n, j_n]$ is a sequence of pixel indices $[i_0, j_0]$, such that the pixel at $[i_k, i_k]$ is a neighbor of the pixel at $[i_{k+l}, i_{k+l}]$ for all k with 0 ~ k ~ n -

Formatted: Font color: Auto

Formatted: Font color: Auto

Formatted: Font color: Auto

1. If the neighbor relation uses 4-connection, then the path is a 4-path; for 8-connection, the path is an 8-path. Simple examples of these are shown in Figure 7. The set of all 1 pixels in an image is called the *foreground* and is denoted by S.

Connectivity

A pixel *p* in S is said to be *connected* to *q* in S if there is a path from *p* to *q* consisting entirely of pixels of S.

Note that connectivity is an equivalence relation. For any three pixels *p*, *q*, and *r* in S, we have the following properties:

1. Pixel *p* is connected to *p* (reflexivity).
2. If *p* is connected to *q*, then *q* is connected to *p* (commutativity).
3. If *p* is connected to *q* and *q* is connected to *r*, then *p* is connected to *r* (transitivity).

Connected Components

A set of pixels in which each pixel is connected to all other pixels is called a *connected component.*

Background

The set of all connected components of *S* (the complement of S) that have points on the border of an image is called the *background.* All other components of *S* are called *holes.*

Let us consider the simple picture shown in Figure 8.

How many objects and how many holes are in this figure? If we consider 4-connectedness for both foreground and background, there are four objects that are 1 pixel in size and there is one hole. If we use 8-connectedness, then there is one object and no hole. Intuitively, in both cases we have an ambiguous situation. To avoid this and similar awkward situations, different connectedness should be used for objects and backgrounds. If we use 8-connectedness for *S*, then 4- connectedness should be used for *S*.

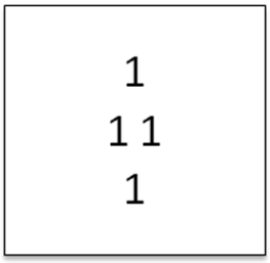


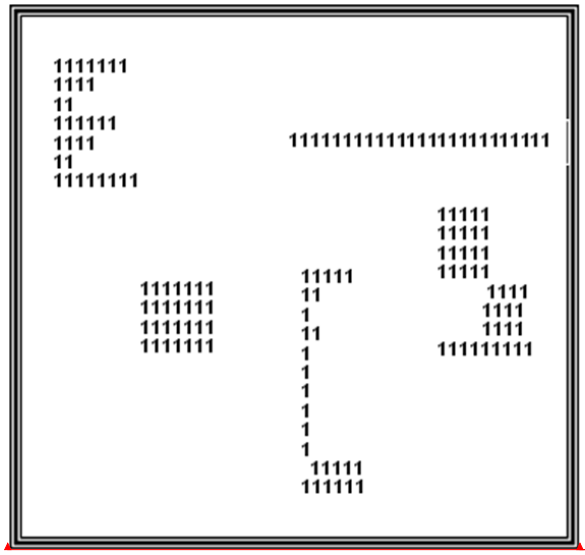
Figure 8. Considering background and foregrounds as different connected component helps.

One of the most common operations in vision systems is finding the connected components in an image. The points in a connected component form a candidate region for representing an object. As mentioned earlier, in computer vision most objects have surfaces. Points belonging to a surface project to spatially close points. The notion of "spatially close" is captured by connected components in digital images.

Formatted: Font color: Auto

Formatted: Font color: Auto

A component labeling algorithm finds all connected components in an image and assigns a unique label to all points in the same component. Figure 9 shows an image and its labeled connected components. In many applications, it is desirable to compute characteristics (such as size, position, orientation, and bounding rectangle) of the components while labeling these components.



Formatted: Font color: Auto

Formatted: Font color: Auto

Figure 9. A binary image with 7 4-connected and 5 8-connected regions (connected components).

Basic Visual Processing Operations: Finding Edges and Regions

Understanding an image requires identifying different objects in it and knowing spatial relationships among them. The understanding of an image is to partition it into clearly marked regions of pixels corresponding to different objects. Such partitions are obtained from the characteristics of the intensity values of the pixels in the image. There are two approaches to partitioning an image into regions: region-based segmentation and boundary estimation using edge detection. Ideally, a region should be bounded by a closed contour, but in many cases using current techniques this is not the case. In principle, region segmentation and edge detection should yield complementary results. Unfortunately, in real images due to noise and other factors, neither region segmentation nor edge detection provides perfect information.

In the next section, we describe basic concepts and techniques for region and edge detection. This is an active area of research. Our goal here is to provide understanding of basic concepts related to region finding and edge detection.

Regions

Pixels must be assigned to regions using criteria that distinguishes them from the rest of the image and helps in separating objects from each other and from the background. Two very important principles in segmentation are *similarity* and *spatial proximity.* Two pixels may be assigned to the same region if they have similar characteristics and if they are close to one another. For example, a specific measure of similarity between two pixels is the difference between the intensity values, and a specific measure of spatial proximity is Euclidean distance. The principles of similarity and proximity are derived from the knowledge that points on the same object will usually project to pixels in the image that are spatially close and have similar intensity values. We can group pixels in an image using these simple assumptions and then use domain-dependent knowledge to match regions to object models. In simple situations, segmentation can be done with thresholding and component labeling, complex images may require more sophisticated techniques to assign pixels to regions that correspond to parts of objects.

Region Segmentation

Let's discuss the process of region formation, or segmentation, more precisely. Given a set of image pixels *I* and a homogeneity predicate *P(.),* find a partition *S* of the image *I* into a set of *n* regions,

$$\bigcup_{i=1}^{n} R_i = I.$$

The homogeneity predicate and partitioning of the image have the properties that any region satisfies the predicate

P(~) = True for all R_i, and any two adjacent regions cannot be merged into a single region that satisfies the predicate

$P(\sim \bigcup R_j)$ = False.

The homogeneity predicate *P(.)* defines the conformity of all points in the region to the region model.

The process of converting an image into a binary image is a simple form of segmentation where the image is partitioned into two sets corresponding to objects and background. The algorithms for thresholding to obtain binary images can be generalized to more than two levels. To make segmentation robust to variations in the scene, the algorithm should be able to select an appropriate threshold automatically by analyzing image intensity values in the image. The knowledge about the intensity values of objects should not be hard-wired into an algorithm; the algorithm should use knowledge about the relative characteristics of intensity values to select the appropriate threshold. This simple idea is

useful in many computer vision algorithms.

Edges

An edge point represents a point that separates two regions corresponding to two different objects. Thus, an edge in an image is a significant local change in the image intensity. Algorithmically, this is usually associated with a discontinuity in either the image intensity or the first derivative of the image intensity. Discontinuities in the image intensity can be either (1) *step* discontinuities, where the image intensity abruptly changes from one value on one side of the discontinuity to a different value on the opposite side, or (2) *line* discontinuities, where the image intensity abruptly changes value but then returns to the starting value within some short distance. However, step and line edges are rare in real images. Because of low-frequency components or the smoothing introduced by most sensing devices, sharp discontinuities rarely exist in real signals. Step edges become *ramp* edges and line edges become *roof* edges, where intensity changes are not instantaneous but occur over a finite distance. Most edges in images are combination of step and line discontinuities. In computer vision, many different approaches have been developed to deal with complex images. Here we will discuss concepts related to edge detection using a general step model.

It is important to define some terms before we discuss edge detection operators to understand precisely what they do and how their results should be analyzed.

An *edge point* is a point in an image with coordinates [i,j] at the location of a significant local intensity change in the image. An *edge fragment* corresponds to the i and j coordinates of an edge and the *edge orientation* e, which may be the gradient angle. An *edge detector* is an algorithm that produces a set of edges {edge points or edge fragments} from an image. A *contour* is a list of edges or the mathematical curve that models the list of edges. *Edge linking* is the process of forming an ordered list of edges from an unordered list. By convention, edges are ordered by traversal in a clockwise direction. Edge following is the process of searching the filtered image to determine contours.

Edge detection is essentially the operation of detecting significant local changes in an image. In one dimension, a step edge is associated with a local peak in the first derivative. The gradient is a measure of change in a function, and an image can be considered to be an array of samples of some continuous function of image intensity. By analogy, significant changes in the gray values in an image can be detected by using a discrete approximation to the gradient. The gradient is the two-dimensional equivalent of the first derivative and is defined as a *vector*. It is common practice to approximate the gradient magnitude by absolute values. As a very simple approximation one may consider edgeness in directions x and y as

$$E_x(x, y) = F(x + 1, y) - F(x,y)$$
$$E_y(x,y) = F(x, y+1) - F(x, y)$$

Then,the greatest magnitude is in direction

$$\alpha = \tan^{-1} (E_y / E_x)$$

and the edge magnitude is

$$E_m = (Ex^2 + Ey^2)^{½}$$

where the angle α is measured with respect to the *x* axis.
Note that the magnitude of the gradient is actually independent of the direction of the edge. Such operators are called *isotropic operators.*

Algorithms for edge detection contain three steps:
Filtering: Since gradient computation based on intensity values of only two points are susceptible to noise and other vagaries in discrete computations, filtering is commonly used to improve the performance of an edge detector with respect to noise. However, there is a trade-off between edge strength and noise reduction.
Enhancement: In order to facilitate the detection of edges, it is essential to determine changes in intensity in the neighborhood of a point. Enhancement emphasizes pixels where there is a significant change in local intensity values and is usually performed by computing the gradient magnitude.
Detection: We only want points with strong edge content. However, many points in an image have a nonzero value for the gradient, and not all of these points are edges for a particular application. Therefore, some method should be used to determine which points are edge points. Frequently, thresholding provides the criterion used for detection. Figure 10 shows an image and strong edges detected in this image.

Segmentation Using Split and Merge

A simple intensity-based segmentation usually results in too many regions. Even in images where most humans see very clear regions with constant intensity value, the output of a thresholding algorithm may contain many extra regions. The main reasons for this problem are high-frequency noise and a gradual transition between intensity values in different regions. After the initial intensity-based region segmentation, the regions may need to be refined or reformed. Several approaches have been proposed for postprocessing such regions obtained from a simple segmentation approach. Some of these approaches use domain-dependent knowledge, while other approaches use knowledge about the imaging process. The refinement may be done interactively by a person or automatically by a computer. In an automatic system, the segmentation will have to be refined based on object characteristics and general knowledge about the images. Automatic refinement is done using a combination of split and merge operations. Split and merge operations eliminate false boundaries and spurious regions by merging adjacent regions that belong to the same object, and they add missing boundaries by splitting regions that contain parts of different objects. Some possible approaches for refinement include:

Figure 10. An image and edges detected in this image.

- Merge adjacent regions with similar characteristics.
- Remove questionable edges.
- Use topological properties of the regions.
- Use shape information about objects in the scene.
- Use semantic information about the scene.

The first three approaches use only information about image intensity combined with other domain-independent characteristics of regions. The other two use domain dependent knowledge. The literature about the segmentation of Images that usually combine above techniques is vast and ever-growing. Such techniques usually result in segmentation shown in Figure 11. We will point to some appropriate sources in further readings.

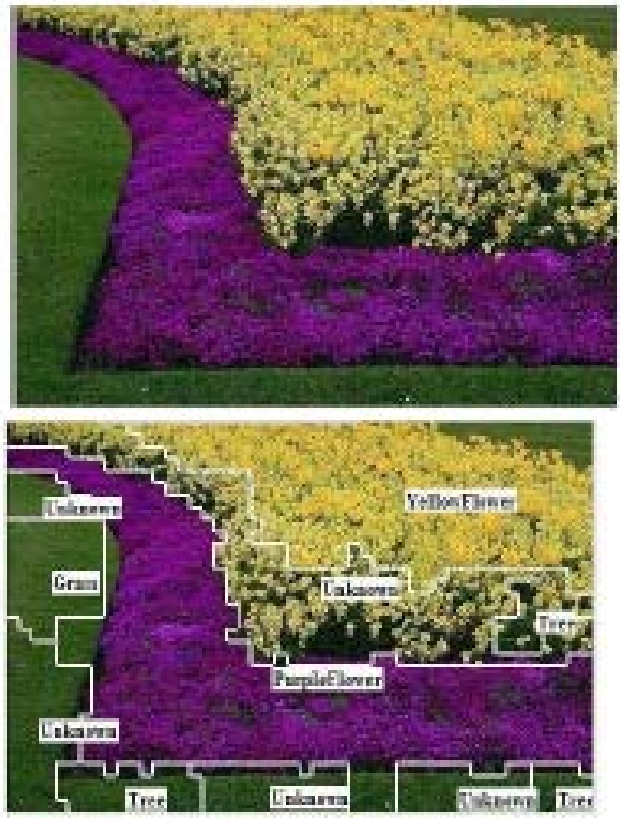


Figure 11. A natural scene image and its segmentation into many known components.

Deleted: XXXXStopped here on Jan 22XXX¶

Prominent Visual Attributes

Humans use some common visual attributes to describe and understand objects and concepts in images. These attributes have been widely studied by psychologists to understand how human visual processing works. These techniques have been widely studied and applied in image understanding and image retrieval systems. In this section, we study three prominent characteristics commonly used in different applications.

Color

An imaging system may use multiple images, commonly called channels, one for each sensor responding to special frequency of light in the imaging system. In color imagery, there are usually three sensors with spectral responses that cover the red, green, and blue regions of the visible spectrum (RGB space). RGB are called primary colors because human visual system is sensitive to these colors and all other colors perceived by humans

are basically a combination of these colors. It is common to work with normalized RGB values, so the set of all possible colors are in a unit cube with corners at (0,0,0) and (1,1,1) in the RGB space. Although RGB are the primary colors, in applications many different transformations are used to suit characteristics of devices and computational requirements.

Hue, brightness, and saturation are considered important characteristics of color based on human perception. Hue is determined by the dominant wavelength in the spectral distribution of light wavelengths. Hue refers to the color names we commonly use. The brightness is a measure of the overall amount of light contained in all three color components. We can think of the brightness as the magnitude of the light as perceived. One measures value or energy at a particular point as brightness or darkness of the pixel. The saturation refers to the dominance of hue in the color. In a way it refers to whether a particular color dominates or the pixel is balanced combination of colors. The pixels that are closer to a specific color, say red, have higher saturation value. In a color wheel commonly used to select colors in many applications, the angle and saturation are selected on the color wheel as shown in the Figure 12 and the intensity or value is selected using a slider. This figure shows how hue and saturation are related to the colors we perceive.

The HSI (or HSV) color model represents a color in terms of hue, saturation, and intensity. This model has been very popular in many applications of image processing. However, there are many other models that are the result of similar transformation of colors by combining the primary colors in different ways that have been defined for specific applications. The RGB components of an image can be converted to the HSI color representation. There are many converters available on line. Most of the color pickers used in word processing and presentation systems use HSV representation rather than RGB representation in their selection palette because of its easy understanding by human beings.

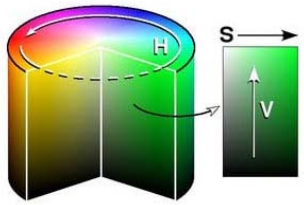


Figure 12. HSV color space. (From: http://coecsl.ece.illinois.edu/ge423/spring05/group8/FinalProject/HSV_writeup.pdf, May be replaced in the final version.)

Deleted: a

Deleted:

Deleted: 0

Formatted: Font: Not Bold

Formatted: Font: Not Bold

Formatted: Font: Not Bold

Formatted: Hyphenate, Don't adjust space between Latin and Asian text, Don't adjust space between Asian text and numbers

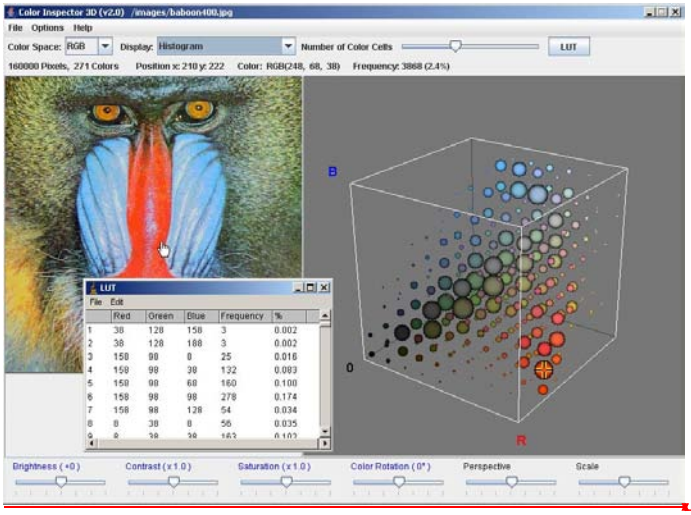


Figure 13. Forming Color Histograms. (Figure may be replaced in the final version.)

Color Histograms

In many applications, color histograms are used to represent either an object or an image. The color histogram is used to represent variations in color in the region. We discussed intensity value color histograms earlier in this chapter, where we collected number of pixels with a particular value in the image. Since a color pixel has three values corresponding to the three color components, the histogram may be a three-dimensional histogram. If each color is represented using a 256 values then one needs a histogram with 256 X 256 X 256 bins. To simplify computations, one may consider the intensity ranges for each component to be represented as 4 bins such that bins 0, 1, 2, and 3 contain ranges of values corresponding to (0 to 63), (64 to 127), (128 to 191), and (192 to 255), respectively. This will reduce the total color space to 64 bins (4 X 4 X4). One can prepare and represent an image using such a histogram. These steps are shown in Figure 13.

The main use of color histograms is to compare one image with other image based on the color distribution in these images. Suppose one is given two images I_1 and I_2 and their histograms are H_1 and H_2. To compare these images one can use a distance measure between the two histograms. Two popular distance measures are the well known L_1 and L_2 distance measures. These result in the distance between two images based on their histograms as,

L_1 distance:

$$D_{H12} = \Sigma_i \, | H_1(i) - H_2(i) |$$

and L_2 distance:

$$D_{HK} = \Sigma_i \, (H_1(i) - H_2(i))^2$$

Deleted: ¶
¶
The HSI (or HSV) color model represents a color in terms of hue, saturation, and intensity. This model has been very popular in many applications of image processing. However, there are many other models that are the result of similar transformation of colors by combining the primary colors in different ways that have been defined for specific applications.¶
¶
¶
The RGB components of an image can be converted to the HSI color representation. Assume that the RGB components have been normalized to 1. This allows the derivation to be done in device-independent units. The intensity is the sum of the RGB values normalized to 1:¶
¶
I = 3(R + G + B)¶
¶
Figure 10. HSV color space.¶
¶
¶

Formatted: Font: Font color: Auto

Deleted: 1

Formatted: Font: Not Bold, Font color: Red

Deleted: 1

One may also consider quadratic distance between two histograms. Consider that each histogram is a vector and consider that there is a matrix C (N X N) that gives similarity of each color i with color j. Then we define quadratic distance between two histograms as:

$$D_Q (H_1,H_2) = (H_1 - H_2)^t \quad C (H_1 - H_2)$$

As is clear, this formulation allows us to consider similarity of individual colors and hence some subjective elements may be introduced easily in this formulation.

Texture

Texture is characterized by the spatial distribution of gray levels in a neighborhood. Thus, texture cannot be defined for a point. The resolution at which an image is observed determines the scale at which the texture is perceived. For example, in observing an image of a tiled floor from a large distance we observe the texture formed by the placement of tiles, but the patterns within the tiles are not perceived. When the same scene is observed from a closer distance, so that only a few tiles are within the field of view, we begin to perceive the texture formed by the placement of detailed patterns composing each tile. For our purposes, we can define texture as repeating patterns of local variations in image intensity, which are too fine to be distinguished as separate objects at the observed resolution. Thus, a connected set of pixels satisfying a given gray-level properties which occur repeatedly in an image region constitutes a textured region. A simple example is a repeated pattern of dots on a white background. Some prominent textures are shown in Figure 14.

Deleted: 2

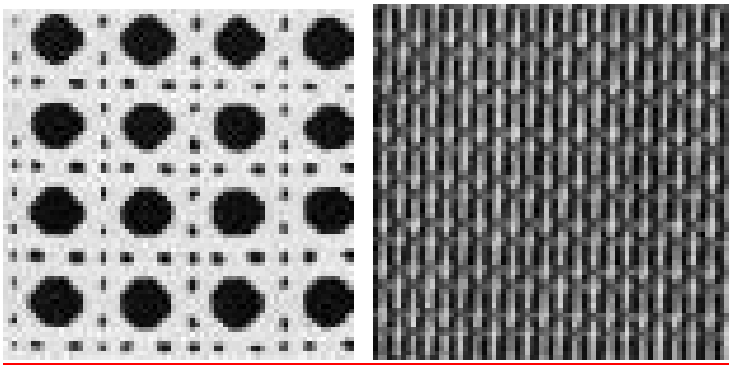


Figure 14. Prominent textures. The first two show regular structure, while the third one shows a texture that is characterized by its statistical characteristics.

Deleted: 2

One common approach to measure and characterize texture is commonly called as Tamura textures. This approach defines six textural features: coarseness, contrast, directionality, line-likeness, regularity and roughness. These features correspond well to what humans usually use to characterize textures and have become popular.

Coarseness has a direct relationship to scale and repetition rates and is considered as the most fundamental texture feature. An image contains textures at several scales; coarseness aims to identify the largest size at which a texture exists, even where a smaller micro texture exists.

Contrast aims to capture the dynamic range of grey levels in an image, together with the polarisation of the distribution of black and white.

Directionality is a global property over a region. The feature described does not aim to differentiate between different orientations or patterns, but measures the total degree of directionality.

Tamura Image is a notion where we calculate a value for the three features at each pixel and treat these as a spatial joint coarseness-contrast-directionality (CND) distribution, in the same way as images can be viewed as spatial joint RGB distributions. We extract color histogram style features from the Tamura CND image, both marginal and 3D histograms. The regional nature of texture meant that the values at each pixel were computed over a window.

Video processing: Analyzing Dynamic Scenes

The input to a dynamic scene analysis system is a sequence of image frames taken from a changing world. The camera used to acquire the image sequence may also be in motion. Each frame represents an image of the scene at a particular instant in time. The changes in a scene may be due to the motion of the camera, the motion of objects, illumination changes, or changes in the structure, size, or shape of an object. It is usually assumed that the changes in a scene are due to camera and/or object motion, and that the objects are either rigid or quasi-rigid. The system must detect changes, determine the motion characteristics of the observer and the objects, characterize the motion using high-level abstraction, recover the structure of the objects, and recognize moving objects. **In** applications such as video editing and video databases, it may be required to detect *macro* changes in a sequence. These changes will partition the segment into many related segments exhibiting similar camera motion or a similar scene in a sequence.
A scene usually contains several objects. An image of the scene at a given time represents a projection of the scene, which depends on the position of the camera. There are four possibilities for the dynamic nature of the camera and world setup:

1. Stationary camera, stationary objects (SCSO)
2. Stationary camera, moving objects (SCMO)
3. Moving camera, stationary objects (MCSO)
4. Moving camera, moving objects (MCMO)

For analyzing image sequences, different techniques are required in each of the above cases. The first case is simply static-scene analysis. Many applications require information extracted from a dynamic environment; in some cases a vision system must understand a dynamic process from a single viewpoint. In applications such as mobile robots or autonomous vehicles, a vision system must analyze an image sequence acquired while in motion. Recovery of information from a mobile camera requires different techniques than those when the camera remains stationary. A sequence of image frames offers much more information to aid in understanding a scene but significantly increases the amount of data to be processed by the system. However, research in dynamic-scene analysis has shown that the recovery of information in many cases is easier in dynamic scenes than in static scenes. In dynamic-scene analysis, SCMO scenes have received the most attention. In analyzing such scenes, the goal is usually to detect motion, to extract masks of moving objects for recognizing them, and to compute their motion characteristics. MCSO and MCMO scenes are very important in navigation applications. MCMO is the most general and possibly the most difficult situation in dynamic scene analysis, but it is also the least developed area of computer vision.

Change Detection

Detection of changes in two successive frames of a sequence is a very important first step for many applications. Any perceptible motion in a scene results in some change in the sequence of frames of the scene. Motion characteristics can be analyzed if such changes are detected. A good quantitative estimate of the motion components of an object may be

obtained if the motion is restricted to a plane that is parallel to the image plane; for three-dimensional motion, only qualitative estimates are possible. Any illumination change in a scene will also result in changes in intensity values, as will scene changes in a TV broadcast or a movie. Most techniques for dynamic-scene analysis are based on the detection of changes in a frame sequence. Starting with frame-to-frame changes, a global analysis of the sequence may be performed. Changes can be detected at different levels: pixel, edge, or region. Changes detected at the pixel level can be aggregated to obtain useful information with which the computational requirements of later phases can be constrained. We discuss different techniques for change detection. We will discuss one of the simplest, yet one of the most useful change detection techniques, *difference pictures*. In a special case, this technique is called background subtraction, when one of the frame is known to be the background because in that case the resulting difference pictures show areas corresponding to moving objects.

Difference Pictures

The most obvious method of detecting change between two frames is to directly compare the corresponding pixels of the two frames to determine whether they are the same. In the simplest form, a binary difference picture *DPjk*(x, y) between frames *F*(x, y, j) and *F*(x, y, k) is obtained by:

$$\begin{aligned} \text{DPjk}(x, y) &= 1 \text{ if } IF(x,y,j) - F(x,y,k)1 > T \\ &= 0 \text{ otherwise} \end{aligned}$$

where *T* is a threshold.

In a difference picture, pixels which have value 1 may be considered to be the result of object motion or illumination changes. This assumes that the frames are properly registered.

A straightforward domain-independent method for comparing regions in images is to consider corresponding areas of the frames. These corresponding areas may be the super-pixels formed by pixels in non-overlapping rectangular areas comprising m rows and n columns. The values of m and n are selected to compensate for the aspect ratio of the camera. A frame maybe partitioned into disjoint super-pixels or use a local mask and compare the intensity distributions. One such method is based on comparing the frames using the likelihood ratio computing over such super-pixels or windows. Thus, we may compute

$$\Lambda = \frac{[(\sigma_1^2 + \sigma_2^2)/2 + (\mu_1 + \mu_2)/2]^2}{\sigma_1^2 + \sigma_2^2}$$

(where μ and σ^2 denote the mean gray value and the variance for the sample areas from the frames, and then use

$$\text{DPjk}(x, y) = 1 \text{ if } \Lambda > T$$

= 0 otherwise.

.

The likelihood ratio test works quite well for most real world scenes. In Figure 13, we show two frames of a sequence and the difference pictures for those. This figure shows the first frame in which it is known that none of the moving objects are there and the second frame is a normal frame. As can be seen, by considering the first frame as the one containing only the static components or the background, a simple subtraction mechanism, commonly called background subtraction, gives the shape and location of all moving objects.

(a) A frame showing background in a scene.

(b) A later frame that has some moving objects.

(c) A difference of frame a and b shows the masks of moving objects.

Figure 13. Background subtraction results in detection of only moving objects.

Computing Image Flow

Image flow, commonly called optical flow, is the velocity field in the image plane due to the motion of the observer, the motion of objects in the scene, or apparent motion which is a change in the image intensity between frames that mimics object or observer motion. Image flow carries valuable information for analyzing dynamic scenes. Several approaches for dynamic-scene analysis have been proposed which assume that image flow information is available. Image flow is determined by the velocity vector of each pixel in an image. Several computational approaches have been devised for calculating image flow based on two or more frames of a sequence. These computational approaches can be classified into two general categories: feature-based and gradient-based. If a stationary camera is used, most of the points in an image frame will have zero velocity. This is assuming that a very small subset of the scene is in motion, which is usually true. Thus, most applications for image flow involve a moving camera.

Although image flow has received a significant amount of attention from researchers, the computing techniques developed for image flow do not produce detailed results of the quality which will allow the valuable information to be recovered. Image flow computations have found applications in compression techniques where one does not require the flow vectors at the same precision level as for recovering dynamic information.

Tracking

In many applications, an object must be tracked over a sequence of frames. If there is only one object in the sequence, the problem is easy to solve. In the presence of many objects moving independently in a scene, tracking requires the use of constraints based on the nature of objects and their motion. Due to inertia, the motion of a physical entity cannot change instantaneously. If a frame sequence is acquired at a rate such that no dramatic change takes place between two consecutive frames, then for most physical objects, no abrupt change in motion can be observed. The projection of a smooth three-

dimensional trajectory is also smooth in the two-dimensional image plane. This allows us to make the smoothness assumption in images. This property is used to formulate *path coherence.* Path coherence implies that the motion of an object at any point in a frame sequence will not change abruptly. We can combine the solution of the correspondence problem for stereopsis and motion. The following three assumptions help in formulating an approach to solve the correspondence problem:

- The location of the given point will be relatively unchanged from one frame to the next frame.
- The scalar velocity of a given point will be relatively unchanged from one frame to the next.
- The direction of motion of a given point will be relatively unchanged from one frame to the next frame.

We can also use the smoothness of image motion in monocular image sequences. Based on such assumption, many computational approaches have been developed to track object or points on objects in videos. This is a very important research area and research has resulted in a rich set of techniques for different applications.

Further Readings

<Incomplete at this time>

Nagel [174] proposed the use of the likelihood ratio for motion detection. Much of the work presented here on difference and accumulative difference pictures was done by Jain with other researchers [129, 130, 122, 125].

Tamura et al took the approach of devising texture features that correspond to human visual perception [1]. Statistical features of grey levels were one of the earliest methods used to classify textures. Haralick [7] suggested the use of grey level co-occurrence matrices (GLCM) to extract second order statistics from an image. GLCMs have been used very successfully for texture classification in evaluations [2]. Turner [9] first implemented this by using a bank of Gabor filters to analyze texture. A bank of filters at different scales and orientations allows multichannel filtering of an image to extract frequency and orientation information. This can then be used to decompose the image into texture features. Our implementation is based on that of Manjunath et al [10,11].